Construction d'un joueur artificiel pour Tetris

Christophe Thiery — **Bruno Scherrer**

LORIA - INRIA Lorraine
Campus Scientifique BP 239
54506 Vandœuvre-lès-Nancy CEDEX
{thierych,scherrer}@loria.fr

RÉSUMÉ. *Nous étudions la conception d'un joueur artificiel pour le jeu de Tetris. Après une revue des principaux travaux, nous soulignons le fait que comparer différentes performances doit être fait avec le plus grand soin, car les scores ont une grande variance, et de subtils détails d'implémentation ont un effet significatif sur les résultats. Nous considérons ensuite la méthode d'entropie croisée pour optimiser la fonction d'évaluation d'un joueur artificiel, comme suggéré par Szita* et al. *(2006). Dans ce contexte, nous discutons de l'influence du paramètre bruit, et nous effectuons des expériences avec plusieurs jeux de fonctions de base, comme celles introduites par Bertsekas* et al. *(1996), par Dellacherie (Fahey, 2003) et des fonctions originales. Cette approche aboutit à un programme de Tetris dont les performances dépassent celles des autres programmes connus. Sur une version simplifiée de Tetris, considérée par la plupart des travaux de recherche, il réalise* $35\,000\,000 \pm 20\,\%$ *de lignes en moyenne par partie.*

ABSTRACT. *We consider the design of a bot for the Tetris game. After a review of the literature, we emphasize the fact that comparing the performances must be done with great care, as the game scores have a huge standard deviation, and as subtle implementation details have a significant effect on the resulting performance. We then consider the cross-entropy method to tune a rating-based one-piece bot as suggested by Szita* et al. *(2006). In this context, we discuss the influence of the noise, and we make experiments with several sets of features such as those introduced by Bertsekas* et al. *(1996), by Dellacherie (Fahey, 2003) and some original ones. This approach leads to a bot that outperforms the previous known results. On a simplified version of Tetris considered by most research works, it achieves* $35,000,000 \pm 20\,\%$ *lines per game on average.*

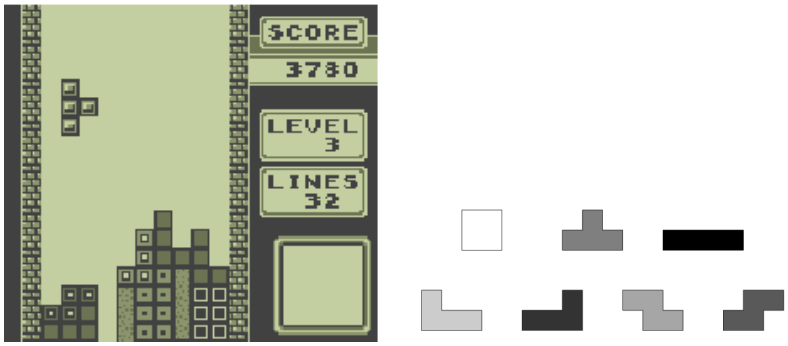
MOTS-CLÉS : *Tetris, méthode d'entropie croisée, apprentissage par renforcement, fonction d'évaluation, fonctions de base*

KEYWORDS: *Tetris, cross-entropy method, reinforcement learning, evaluation fonction, features*

RSTI - RIA – 23/2009. Jeux : modélisation et décision, pages 387 à 407

1. Introduction

Tetris est un célèbre jeu vidéo créé en 1985 par Alexey Pajitnov. Le jeu se déroule sur une grille 2D de taille 10×20 (10 colonnes et 20 lignes), où des pièces de différentes formes tombent du haut de la grille les unes après les autres (voir figure 1). Le joueur doit choisir où il place chaque pièce : il peut déplacer la pièce horizontalement et la faire pivoter. Lorsqu'une ligne est pleine, celle-ci est supprimée et toutes les cellules au-dessus d'elle descendent d'une ligne. L'objectif est de supprimer un maximum de lignes avant que la partie ne soit terminée. La partie se termine lorsqu'il ne reste plus assez d'espace libre en haut de la grille pour placer la pièce courante. On peut trouver une spécification détaillée de Tetris sur le site de Fahey (Fahey, 2003).



(a) (b)

Figure 1. *Une image du jeu de Tetris (a). Les sept pièces possibles (b)*

Tetris a fait l'objet de plusieurs travaux de recherche (la section 2 en donne un aperçu). C'est un jeu plutôt difficile : il contient un grand nombre de configurations (de l'ordre de $2^{200} \simeq 10^{60}$). De plus, on sait aujourd'hui que trouver une séquence de coups qui maximise le nombre de lignes est un problème NP-complet, même dans le cas où la séquence de pièces est connue à l'avance (Demaine *et al.*, 2003).

Pour comparer la qualité de plusieurs joueurs artificiels à Tetris, il est important de définir une mesure de performance. Comme Tetris est un jeu stochastique, une mesure naturelle, considérée par tous les travaux dont nous avons connaissance, est le nombre moyen de lignes réalisées avant de perdre la partie. Cette mesure est toujours finie : il a été montré que toute partie de Tetris se termine avec probabilité 1 (Burgiel, 1997).

Comme la plupart des chercheurs, nous nous intéressons à une version simplifiée de Tetris. D'abord, nous considérons uniquement les *contrôleurs à une pièce*, c'est-à-dire ceux qui connaissent la configuration de la grille et la pièce courante uniquement. Un joueur artificiel qui connaît également la forme de la prochaine pièce, comme cela est le cas dans le Tetris original, est appelé un *contrôleur à deux pièces*. Un tel joueur est avantagé puisqu'il peut tirer parti de cette connaissance supplémentaire pour prendre de meilleures décisions. Il est assez immédiat d'étendre des travaux sur des contrôleurs à une pièce pour construire des contrôleurs à deux pièces. Des données

expérimentales suggèrent que les performances d'un joueur artificiel sont ainsi améliorées de plusieurs ordres de grandeur (Fahey, 2003). Dans un souci de simplification, nous considérerons dans cet article uniquement des contrôleurs à une pièce.

Dans le jeu original de Tetris (cf. (Fahey, 2003)), la pièce courante apparaît *à l'intérieur* de la zone de jeu, puis descend graduellement. La partie est perdue lorsque la pièce n'a pas assez d'espace pour apparaître en haut de la grille. Comme dans la majorité des travaux (Tsitsiklis *et al.*, 1996; Bertsekas *et al.*, 1996; Kakade, 2001; Lagoudakis *et al.*, 2002; Ramon *et al.*, 2004; Farias *et al.*, 2006; Szita *et al.*, 2006), nous considérons une simplification de Tetris : le joueur artificiel décide simplement dans quelle colonne et avec quelle orientation il lâche la pièce. De cette manière, le jeu est légèrement simplifié car la pièce apparaît dans la zone de jeu directement là où le joueur a décidé de la placer. Ainsi, on considère qu'il y a toujours de l'espace au-dessus de la zone de jeu pour choisir l'orientation et la colonne où l'on va lâcher la pièce. Cela produit une différence importante, car l'intégralité de l'espace de la grille devient utilisable, y compris les lignes les plus hautes. On évite ainsi les situations où la hauteur du mur empêcherait la pièce d'atteindre un côté de la grille. Ce jeu de Tetris simplifié est donc plus facile que le jeu original, et un joueur artificiel a la possibilité de réaliser un plus grand nombre de lignes.

Pour un joueur humain, l'une des difficultés de Tetris réside dans le fait que les pièces tombent relativement rapidement du haut de la zone de jeu : le peu de temps alloué rend parfois difficile la prise de décision. Cette dimension du problème n'apparaît pas quand il s'agit de construire un joueur artificiel. Les joueurs artificiels que nous étudions dans la suite de cet article sont capables de jouer 50 000 à 100 000 coups par seconde sur une machine de bureau actuelle. La vitesse de chute des pièces est donc négligeable par rapport à la vitesse à laquelle un joueur artificiel est capable de prendre une décision. Comme dans la majorité des travaux sur Tetris (Tsitsiklis *et al.*, 1996; Bertsekas *et al.*, 1996; Kakade, 2001; Lagoudakis *et al.*, 2002; Ramon *et al.*, 2004; Farias *et al.*, 2006; Szita *et al.*, 2006), nous allons ignorer la chute des pièces et nous concentrer uniquement sur le cœur du problème, qui est de décider où placer chaque pièce qui se présente.

La suite de cet article est organisée de la manière suivante. La section 2 mentionne les précédents travaux qui ont essayé de construire des joueurs artificiels pour Tetris ; en particulier, nous détaillons le meilleur joueur artificiel connu, qui a été proposé par Dellacherie (Fahey, 2003). La section 3 souligne d'importants problèmes qui se posent lorsque l'on essaie de comparer différents joueurs artificiels. Dans la section 4, nous considérons le récent travail de Szita et Lőrincz utilisant une méthode d'entropie croisée, et nous approfondissons son analyse empirique. Enfin, la section 5 décrit comment nous avons combiné les fonctions de base de Dellacherie et la méthode de Szita et Lőrincz pour obtenir un joueur artificiel à une pièce dont les performances dépassent celles des précédentes approches.

2. Etat de l'art

Dans la littérature, tous les travaux visant à concevoir des joueurs artificiels pour Tetris se fondent sur une *fonction d'évaluation*. Dans un état donné du jeu (c'est-à-dire la configuration actuelle du mur et la pièce courante), toutes les *actions* possibles (c'est-à-dire le choix d'une position et d'une orientation pour la pièce courante) sont évaluées à l'aide de cette fonction (voir figure 2). Le joueur artificiel choisit alors l'action qui a reçu la meilleure évaluation.

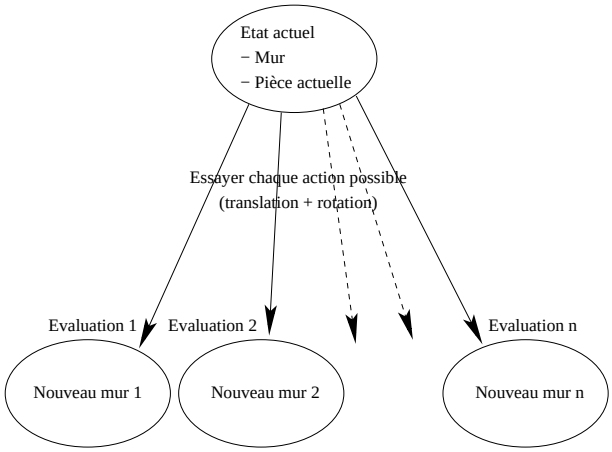


Figure 2. *Principe d'un contrôleur à une pièce*

Ainsi, le problème de créer un joueur de Tetris revient à concevoir une bonne fonction d'évaluation. Idéalement, on voudrait que cette fonction retourne des valeurs élevées pour les bonnes décisions, et de faibles valeurs pour les mauvaises. L'évaluation est une combinaison de *fonctions de base*, typiquement une somme pondérée (mais pas toujours, voir par exemple (Böhm *et al.*, 2005)). Les fonctions de base tentent de capturer les caractéristiques pertinentes des actions et des états. Un exemple de fonction de base à Tetris est le nombre de trous[1] dans le mur résultant d'une décision. Son poids associé est habituellement négatif : plus il y a de trous, plus l'évaluation est basse car les trous empêchent de faire des lignes.

Concevoir un joueur artificiel pour Tetris basé sur une fonction d'évaluation se fait généralement en deux étapes. La première étape consiste à choisir un ensemble de fonctions de base approprié, c'est-à-dire capable d'extraire les informations pertinentes du jeu, et elle est habituellement accomplie par un expert. La seconde étape consiste à déterminer le poids de chaque fonction de base dans la somme pondérée. Dans la littérature, cette étape est faite manuellement (par un expert) ou automatique-

1. Un trou est une cellule vide recouverte par une cellule pleine.

ment (par apprentissage par renforcement ou par des techniques d'optimisation). Nous résumons ci-dessous les résultats les plus significatifs à notre connaissance.

– **Approches par apprentissage par renforcement.** Plusieurs travaux de la littérature d'apprentissage par renforcement (Sutton *et al.*, 1998) considèrent le jeu de Tetris ; des algorithmes visent à fixer les poids de manière à ce que la fonction d'évaluation approxime l'espérance optimale du score futur à partir de chaque état. La première approche concernant Tetris dans ce domaine semble être dûe à Tsitsiklis et Van Roy (Tsitsiklis *et al.*, 1996) ; le joueur obtenu utilise deux fonctions de base et réalise un faible score moyen (environ une trentaine de lignes). Plus tard, Bertsekas et Tsitsiklis ont proposé l'algorithme λ-*Policy Iteration* et l'ont appliqué à Tetris (Bertsekas *et al.*, 1996). Pour ce faire, ils ont proposé un ensemble de fonctions de base que nous décrivons dans le tableau 1. Leur approche permet d'atteindre un score moyen de 3 200 lignes sur 100 parties jouées. Cet ensemble de fonctions de base a été réutilisé par la suite dans deux autres travaux d'apprentissage par renforcement : 1) Kakade a appliqué la méthode *natural policy gradient* (Kakade, 2001) et a mesuré un score moyen d'environ 6 800 lignes par partie (sans cependant spécifier sur combien de parties cette moyenne a été réalisée) ; 2) Farias et Van Roy ont appliqué une approche de programmation linéaire (Farias *et al.*, 2006), atteignant un score moyen de 4 700 lignes sur 90 parties jouées. Lagoudakis a appliqué la méthode *Least-Squares Policy Iteration* avec des fonctions de base originales (Lagoudakis *et al.*, 2002), mesurant un score moyen de 1 000 à 3 000 lignes. Même s'il n'a pas abouti à des performances élevées, nous pouvons également mentionner le travail de Ramon et Driessens (Ramon *et al.*, 2004) basé sur de l'apprentissage par renforcement relationnel.

Fonction de base	**Id**	**Description**	**Commentaires**
Hauteur de colonne	h_p	Hauteur de la p-ième colonne du mur	P fonctions de bases, où P est la largeur de la grille
Différence inter colonne	Δh_p	Différence en valeur absolue entre deux colonnes adjacentes : $\lvert h_p - h_{p+1} \rvert$	$P-1$ fonctions de base, où P est la largeur du mur
Hauteur maximale	H	Hauteur maximale du mur : $\max_p h_p$	Empêche d'augmenter la hauteur du mur
Trous	L	Nombre de cases vides recouvertes par au moins une case pleine	Empêche de faire des trous

Tableau 1. *Fonctions de base proposées par Bertsekas et Tsitsiklis*

– **Approches d'optimisation générale.** Une alternative à l'apprentissage par renforcement pour déterminer les poids est d'utiliser des techniques d'optimisation générale, où un algorithme cherche directement des poids tels que le joueur artificiel correspondant soit performant. Contrairement à l'apprentissage par renforcement où la fonction d'évaluation est une approximation du score futur, la fonction d'évaluation n'a ici pas de sémantique. Par exemple, le programme de l'implémentation GNU

Xtris utilise 6 fonctions de base dont les poids ont été optimisés grâce à des algorithmes génétiques (Llima, 2005) : sur un simulateur très proche du Tetris original, ce programme réalise une moyenne de $50\ 000$ lignes par partie. Böhm *et al.* (2005) reportent également de bons résultats avec une approche évolutionnaire, mais ils considèrent uniquement des contrôleurs à deux pièces et ne fournissent pas de résultats précis sur la grille standard de taille 10×20. Récemment, Szita et Lőrincz (Szita *et al.*, 2006) ont appliqué la méthode d'entropie croisée avec les fonctions de base de Bertsekas et Tsitsiklis, et ont obtenu un score moyen de $350\ 000$ lignes par partie (sur 30 parties), dépassant ainsi les approches par apprentissage par renforcement qui utilisent ces mêmes fonctions de base.

– **Joueurs artificiels construits manuellement.** A notre connaissance, le meilleur contrôleur à une pièce à l'heure actuelle est celui de Dellacherie (Fahey, 2003) et a été paramétré à la main. Dellacherie a mis au point un ensemble efficace de fonctions de base et a fixé ses poids manuellement. Etonnamment, ce joueur construit à la main dépasse les performances des travaux d'apprentissage par renforcement et les récents résultats de Szita et Lőrincz : sur un total de 56 parties, l'algorithme de Dellacherie a atteint un score moyen d'environ $660\ 000$ lignes. De plus, il faut noter que cette mesure de $660\ 000$ lignes par partie n'a pas été réalisée avec la version simplifiée usuelle de Tetris décrite plus haut, mais avec une implémentation du Tetris original qui, comme mentionné dans la section 1, est plus difficile. Le code source étant mis à disposition par Colin Fahey (Fahey, 2003), nous l'avons analysé pour déterminer les fonctions de base et leurs poids. La fonction d'évaluation de Dellacherie est la somme pondérée suivante :

$$-l + e - \Delta r - \Delta c - 4L - W$$

où les fonctions de base $l, e, \Delta r, \Delta c, L$ et W sont détaillées dans le tableau 2.

D'une part, les fonctions de base de Dellacherie semblent être les plus compétitives : même avec des poids choisis à la main, le joueur artificiel de Dellacherie donne jusqu'à présent les meilleurs résultats. D'autre part, la méthode d'entropie croisée (Szita *et al.*, 2006) apparaît comme étant l'algorithme le plus performant pour régler les poids d'un ensemble de fonctions de base donné. Le joueur artificiel que nous allons construire dans cet article s'appuie sur ces deux observations : comme décrit dans la section 5, nous allons exploiter les fonctions de base efficaces de Dellacherie et utiliser la méthode d'entropie croisée pour fixer leurs poids.

3. Difficulté de comparer les joueurs artificiels

Avant d'étudier la méthode d'entropie croisée, il est important de noter que comparer des joueurs artificiels pour Tetris est un problème délicat, en particulier lorsque leurs performances sont mesurées sur des implémentations différentes. D'abord, nous expliquons que le score moyen réalisé par un joueur artificiel a une grande variance.

Fonction de base	Id	Description	Commentaires
Hauteur d'arrivée	l	Hauteur où la dernière pièce a été posée	Empêche d'augmenter la hauteur du mur
Erosion	e	(Nombre de lignes réalisées au dernier coup) $\times$ (Nombre de cellules éliminées dans la dernière pièce posée)	Incite à réaliser des lignes
Transitions de lignes	Δr	Nombre de transitions plein/vide ou vide/plein entre les cellules sur chaque ligne	Rend le mur homogène
Transitions de colonnes	Δc	Même chose pour les transitions verticales	
Trous	L	Nombre de cases vides recouvertes par au moins une case pleine	Empêche de faire des trous
Puits	W	$\sum_{w \in puits}(1 + 2 + \cdots + depth(w))$	Empêche de faire des puits[2]

Tableau 2. *Fonctions de base proposées par Dellacherie*

Ensuite, nous montrons que des détails subtils d'implémentation peuvent faire varier significativement les résultats d'un joueur artificiel.

3.1. *Grande variance des scores à Tetris*

Bien que la plupart des auteurs semblent conscients du fait que les scores à Tetris ont une importante variance, presque aucun ne fournit des intervalles de confiance. A notre connaissance, le travail de Szita et Lőrincz (Szita *et al.*, 2006) est le seul à le faire. Nous expliquons ici comment calculer ces intervalles de confiance.

Fahey a conjecturé que le score d'une partie pour un joueur artificiel donné suit une distribution exponentielle (Fahey, 2003). Cette conjecture a été validée expérimentalement (à $95\ \%$) par un test statistique réalisé par Szita et Lőrincz (Szita *et al.*, 2006). Le fait que le score suive une distribution exponentielle permet d'en déduire un intervalle de confiance. En général, si X est une variable aléatoire de moyenne μ et de variance σ^2, le théorème central limite établit que, à $95\ \%$, la moyenne empirique $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} X_i$ (où les X_i sont N réalisations indépendantes de X, pour N suffisamment grand) satisfait :

$$|\mu - \hat{\mu}| \leq \frac{2\sigma}{\sqrt{N}}.$$

2. Un puits est une succession de cases vides dans une colonne, telles que leurs deux cellules voisines à droite et à gauche sont occupées. Les puits profonds sont pénalisants car ils forcent à attendre une barre verticale ou à faire des trous.

Lorsque X suit une distribution exponentielle, on a $\sigma = \mu$ et il est alors usuel d'estimer que :

$$|\mu - \hat{\mu}| \leq \frac{2\mu}{\sqrt{N}} \simeq \frac{2\hat{\mu}}{\sqrt{N}}$$

ce qui nous conduit à l'intervalle de confiance *relatif* suivant :

$$\frac{|\mu - \hat{\mu}|}{\hat{\mu}} \leq \frac{2}{\sqrt{N}}.$$

En considérant l'évaluation du joueur artificiel de Dellacherie sur $N = 56$ parties, on peut conclure de l'analyse précédente que l'intervalle de confiance de la moyenne empirique (660 000 lignes) est de $27\ \%$ avec probabilité $0,95$. Même si l'intervalle de confiance est très grand, le joueur artificiel de Dellacherie reste le meilleur.

La plupart des joueurs artificiels que nous étudierons dans la suite de cet article seront évalués avec $N = 100$ parties. Cela donnera un intervalle de confiance de $\pm 20\ \%$, valide à $95\ \%$. Les intervalles de confiance donnés indiqueront implicitement le nombre d'échantillons utilisés pour l'évaluation. De plus, étant donné que tous les intervalles de confiance que nous donnerons sont valides à $95\ \%$, nous retirerons la mention "valide à 95 %" pour plus de lisibilité.

3.2. *Subtilités d'implémentation*

Nous venons de voir qu'en général, les intervalles de confiance que l'on peut calculer pour des joueurs artificiels pour Tetris sont assez grands. Lorsque nous avons implémenté notre propre simulateur de Tetris, nous avons également remarqué que certains détails subtils dans l'implémentation pouvaient avoir un effet significatif sur les résultats obtenus.

Un premier détail subtil (qui n'est jamais explicité dans les travaux dont nous avons connaissance) est de savoir comment un joueur artificiel utilisant une fonction d'évaluation se comporte lorsqu'il est proche de perdre la partie. Il se peut qu'à un moment donné, la décision qui a reçu la meilleure évaluation fasse perdre la partie immédiatement, alors que d'autres décisions (avec des évaluations plus basses) auraient permis de continuer le jeu plus longtemps. Dans un tel cas, il est préférable de ne pas considérer comme des actions possibles celles qui mènent directement à la fin de la partie : la partie dure alors plus longtemps et le score sera meilleur. Si l'on choisit de procéder de cette manière, cela signifie que la partie est perdue si et seulement si *toutes* les décisions font perdre la partie. C'est le choix que nous avons fait. Sur notre simulateur, le joueur artificiel de Dellacherie réalise ainsi $5\ 200\ 000 \pm 20\ \%$ lignes en moyenne. Si nous laissons le joueur artificiel effectuer des actions qui font perdre (car elles ont la meilleure évaluation), les résultats de Dellacherie tombent à $850\ 000 \pm 20\ \%$ lignes.

De plus, la plupart des implémentations de Tetris définissent la fin de la partie comme étant le moment où la pièce courante n'a pas assez d'espace pour être placée

dans la zone de jeu, c'est-à-dire lorsque la pièce déborde de la grille de taille 10×20. C'est la définition que nous avons adoptée. Cependant, si nous examinons de plus près la description de Tetris écrite par Bertsekas et Tsitsiklis (Bertsekas *et al.*, 1996), nous pouvons voir qu'ils considèrent que la partie *"se termine lorsqu'une case de la ligne du haut devient pleine et que le haut du mur atteint le haut de la grille"*. Cette définition est équivalente à dire que la pièce déborde d'une grille de taille 10×19. Ce genre de détail peut produire une différence significative sur les scores : avec une grille de taille 10×19, l'algorithme de Dellacherie réalise $2\ 500\ 000 \pm 20\ \%$ lignes avec notre implémentation au lieu de $5\ 200\ 000 \pm 20\ \%$. Par conséquent, nous pensons que les résultats expérimentaux de Bertsekas et Tsitsiklis, mentionnés dans la section 2, sont sous-estimés par rapport aux autres travaux d'apprentissage par renforcement.

Comme des petits détails concernant les règles du jeu et l'implémentation du joueur artificiel peuvent avoir des effets significatifs sur les scores, il est nécessaire d'employer le plus grand soin lorsque l'on compare différents joueurs artificiels. Le seul moyen d'effectuer une comparaison fiable de plusieurs joueurs artificiels est de les lancer sur le même simulateur et un grand nombre de fois. Pour ce faire, nous avons implémenté un simulateur de Tetris configurable et optimisé, ainsi que plusieurs joueurs artificiels[3].

Dans la suite de cet article, nous comparons les résultats des joueurs artificiels considérés avec les performances du joueur artificiel de Dellacherie sur deux configurations de jeu :

– la première configuration est le jeu de Tetris standard utilisé par la plupart des travaux de recherche. Nous utilisons une grille de taille 10×20 en considérant comme référence le meilleur score que nous connaissons : $5\ 200\ 000 \pm 20\ \%$ lignes en moyenne, qui est le score atteint par le joueur artificiel de Dellacherie sur notre implémentation ;

– la seconde configuration nous permet de déduire une borne inférieure sur le score de nos joueurs artificiels s'ils étaient lancés sur une implémentation du Tetris original tel que spécifié par (Fahey, 2003). Dans le Tetris original, les pièces doivent être déplacées étape par étape jusqu'à leur position finale. Comme expliqué dans la section 1, par rapport au Tetris simplifié, la principale difficulé est qu'il peut y avoir des problèmes de collisions lorsque le mur est trop haut. Comme la hauteur des pièces est inférieure ou égale à 4, il est clair qu'un joueur artificiel capable d'atteindre un certain score sur notre implémentation avec une grille de taille 10×16 fera un meilleur score sur le Tetris original en taille 10×20. Pour cette raison, nous considérerons aussi les performances de nos joueurs artificiels sur un jeu de taille 10×16 et nous comparerons ces résultats avec le score de $660\ 000 \pm 27\ \%$ lignes (obtenu par le joueur artificiel de Dellacherie sur le simulateur de Fahey du Tetris original (Fahey, 2003)). On note cependant que cette borne inférieure est assez pessimiste car, en taille 10×16, notre implémentation du joueur artificiel de Dellacherie ne réalise que $220\ 000 \pm 20\ \%$ lignes.

3. Le code source C est disponible ici : `http://gforge.inria.fr/projects/mdptetris`.

4. L'approche de Szita et Lőrincz

Malgré les réserves que nous venons de faire à propos de la comparaison de joueurs artificiels sur différentes implémentations, l'entropie croisée semble être actuellement l'approche la plus efficace pour régler les poids de la fonction d'évaluation d'un joueur artificiel pour Tetris. Szita et Lőrincz (Szita *et al.*, 2006) ont récemment montré que cette méthode améliore les scores des travaux d'apprentissage par renforcement de plusieurs ordres de grandeur. Dans cette section, nous décrivons la méthode d'entropie croisée et nous expliquons comment Szita et Lőrincz l'ont appliquée à Tetris. Nous reproduisons leur expérience et nous effectuons des observations légèrement différentes sur le paramètre de bruit, qui est crucial pour l'efficacité de l'algorithme.

4.1. *La méthode d'entropie croisée*

Nous présentons d'abord les principes de la méthode d'entropie croisée. Cette description est inspirée de celle de Szita et Lőrincz (Szita *et al.*, 2006). Une description plus détaillée de cet algorithme d'optimisation peut être trouvée dans (de Boer *et al.*, 2004). La méthode d'entropie croisée est un algorithme stochastique itératif qui cherche à résoudre un problème d'optimisation de la forme :

$$w^* = \arg\max_w S(w)$$

où S est une fonction que l'on souhaite maximiser et w est un paramètre à optimiser (typiquement un vecteur).

La méthode d'entropie croisée itère sur une distribution de solutions et non sur une solution seule. On considère une famille de distributions $\mathcal{F}$ (par exemple les distributions gaussiennes) et on veut déterminer une distribution de probabilité $f \in \mathcal{F}$ qui génère des solutions w proches de la solution optimale w^*. A chaque itération t, on a une distribution $f_t \in \mathcal{F}$, et on veut que la distribution suivante $f_{t+1} \in \mathcal{F}$ produise de meilleures solutions. Pour cela, on considère qu'une solution w est une bonne solution si sa valeur est supérieure à un certain seuil γ_t, c'est-à-dire si $S(w) > \gamma_t$. Considérons alors g_{γ_t}, la distribution de probabilité uniforme qui génère des solutions dont les valeurs sont supérieures à γ_t. En général, cette distribution g_{γ_t} n'appartient pas à $\mathcal{F}$. L'idée de la méthode d'entropie croisée est de chercher la distribution $f_{t+1} \in \mathcal{F}$ qui s'en rapproche le plus, au sens de la *mesure d'entropie croisée*[4] (de Boer *et al.*, 2004). Pour de nombreux types de familles de distributions $\mathcal{F}$, cette distribution f_{t+1} peut être estimée à partir d'échantillons générés par la distribution f_t. Par exemple, dans le cas où $\mathcal{F}$ est l'ensemble des distributions gaussiennes, la distribution gaussienne la plus proche de g_{γ_t} est celle qui est caractérisée par la moyenne et la variance de la distribution g_{γ_t}. On peut estimer ces paramètres en générant des échantillons avec la

4. La mesure d'entropie croisée (ou *distance de Kullback-Leibler*) définit une notion de distance entre deux distributions de probabilités.

distribution f_t et en sélectionnant ceux qui sont au-dessus du seuil γ_t, c'est-à-dire les meilleurs.

En pratique, la séquence des seuils γ_t est construite automatiquement en même temps que celle des distributions f_t. Précisément, la méthode d'entropie croisée telle que nous l'utilisons dans cet article est détaillée dans l'algorithme 1, et une représentation graphique est donnée à la figure 3. Globalement, elle consiste à répéter les étapes suivantes :

– on génère N échantillons à partir de la distribution gaussienne actuelle f_t ;

– on évalue chacun de ces N vecteurs vis-à-vis de la fonction à optimiser S ;

– on sélectionne une proportion $\rho \in]0, 1[$ des meilleures solutions (cela revient à fixer γ_t à un certain seuil) ;

– on fixe les paramètres de la nouvelle distribution gaussienne f_{t+1} comme étant la moyenne et la variance empiriques des meilleures solutions sélectionnées.

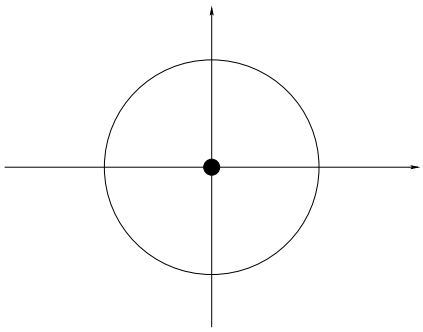
Algorithme 1 Méthode d'entropie croisée bruitée avec une distribution gaussienne

Entrées :
$evaluer()$: une fonction qui estime la fonction à optimiser S pour un certain vecteur w
(μ, σ) : la moyenne et la variance de la distribution initiale
N : le nombre de vecteurs générés à chaque itération
ρ : la fraction de vecteurs sélectionnés
Z_t : le bruit ajouté à chaque itération

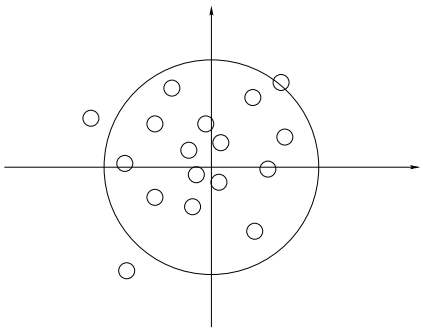
répéter :
Générer N vecteurs $w_1, w_2, \ldots, w_N$ selon $\mathcal{N}(\mu, \sigma^2)$
Evaluer chaque vecteur à l'aide de la fonction $evaluer()$
Sélectionner les $\lfloor \rho \times N \rfloor$ vecteurs ayant reçu les meilleures évaluations
$\mu \leftarrow$ (moyenne des vecteurs sélectionnés)
$\sigma^2 \leftarrow$ (variance des vecteurs sélectionnés) $+Z_t$
fin

Dans la description détaillée de l'algorithme 1, la fonction $evaluer()$ qui est utilisée pour évaluer chaque vecteur peut être S, ou bien une approximation de S si S prend trop de temps pour être calculée de manière exacte. Par ailleurs, à chaque itération, un terme de *bruit* Z_t est ajouté à la mise à jour de la variance. Lorsque $Z_t \neq 0$, l'algorithme est appelé algorithme d'entropie croisée bruitée (de Boer *et al.*, 2004). En pratique, on peut voir ce terme de bruit comme un moyen d'éviter une convergence trop rapide vers un mauvais optimum local.

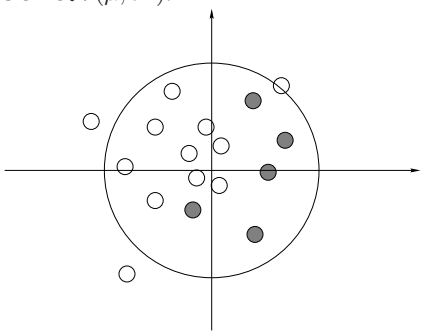
L'algorithme d'entropie croisée est relativement proche des algorithmes évolutionnaires. C'est un processus itératif qui traite un ensemble de solutions candidates (ou individus). A chaque itération, les meilleurs individus sont sélectionnés, puis de nouveaux individus sont générés à partir d'eux. La principale particularité de la méthode d'entropie croisée est la manière dont les nouvelles solutions sont générées.



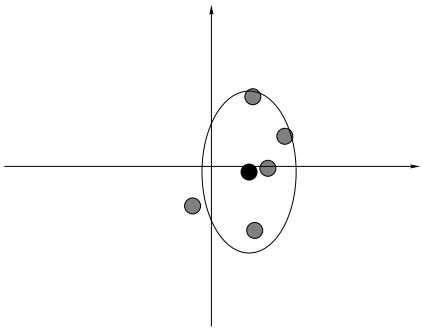
1. Commencer avec une distribution gaussienne $\mathcal{N}(\mu, \sigma^2)$.



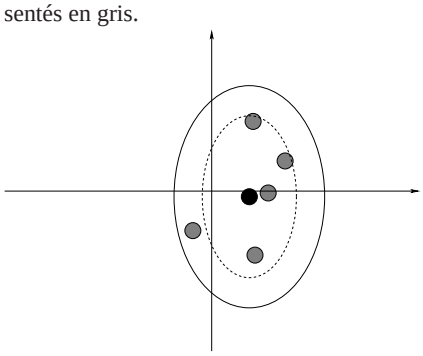
2. Générer N vecteurs avec cette distribution.



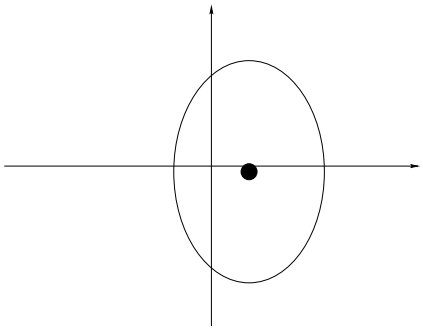
3. Evaluer chaque vecteur avec $evaluer()$ et sélectionner une proportion ρ des meilleurs vecteurs. Ces vecteurs sont représentés en gris.



4. Calculer la moyenne et la variance des meilleurs vecteurs.



5. Ajouter un terme de bruit à la variance, afin d'éviter une convergence trop rapide vers un optimum local.



6. Cette moyenne et cette variance caractérisent une nouvelle distribution gaussienne qui générera des vecteurs pour la prochaine itération.

Figure 3. *Une représentation graphique de la méthode d'entropie croisée bruitée pour optimiser un vecteur à deux dimensions. La distribution gaussienne est representée avec un disque noir pour la moyenne et une ellipse pour la variance*

4.2. *Application à Tetris*

Szita et Lőrincz ont appliqué la méthode d'entropie croisée avec une distribution gaussienne au problème de Tetris (Szita *et al.*, 2006). Ils considèrent un joueur artificiel qui utilise les fonctions de base de Bertsekas et Tsitsiklis, car ce type de joueur artificiel avait déjà été utilisé dans plusieurs travaux (Bertsekas *et al.*, 1996; Kakade, 2001; Farias *et al.*, 2006). Un tel joueur artificiel base ses décisions sur la fonction d'évaluation suivante :

$$\sum_{i=1}^{10} w_i h_i + \sum_{i=1}^{9} w_{10+i} \Delta h_i + w_{20} H + w_{21} L$$

où les fonctions de base $h_i, \Delta h_i, H$ et L sont détaillées dans le tableau 1, et où $w = (w_1, \ldots, w_{21})$ est le vecteur de poids à déterminer. Naturellement pour cette application, la fonction $w \mapsto S(w)$ à optimiser est l'espérance du score atteint par le joueur artificiel correspondant au vecteur de poids w. Szita et Lőrincz partent d'une distribution gaussienne centrée à $\mu = (0, 0, \ldots, 0)$ avec variance $\sigma^2 = (100, 100, \ldots, 100)$. A chaque itération, ils génèrent $N = 100$ vecteurs et évaluent chacun d'entre eux en jouant une partie. Ils sélectionnent les 10 meilleurs vecteurs ($\rho = 10\ \%$) et génèrent ainsi la nouvelle distribution gaussienne. Après chaque itération, ils jouent 30 parties avec les poids moyens de la nouvelle distribution, afin d'obtenir une courbe d'apprentissage représentant l'évolution des performances au fur et à mesure des itérations.

Dans l'expérience de Szita et Lőrincz , la fonction $evaluer()$ est le score d'*une seule* partie. Comme nous avons vu dans la section 3 que les scores à Tetris ont une grande variance, il est clair que cette évaluation n'est pas précise. Avec notre implémentation, nous avons essayé d'évaluer chaque vecteur en jouant plus de parties pour voir si c'était un choix crucial, et nous en avons conclu que ce n'était pas le cas. En effet, même si nous avons observé que le nombre d'itérations nécessaires pour atteindre le niveau de performance maximal est inférieur (ce qui est naturel puisque le processus de sélection est plus précis), nous avons remarqué qu'après convergence, les joueurs artificiels obtenus n'étaient pas meilleurs.

Szita et Lőrincz ont lancé la méthode d'entropie croisée dans trois conditions expérimentales : sans bruit ($Z_t = 0$), avec un bruit constant ($Z_t = 4$), et avec un bruit linéairement décroissant ($Z_t = \max(5 - t/10, 0)$). Les formules des bruits constant et linéairement décroissant ont été fixées à la suite d'expériences préliminaires. Leurs résultats indiquent que les performances sont significativement améliorées lorsque l'on utilise du bruit. Leur meilleur joueur artificiel a été obtenu avec le bruit décroissant, atteignant un score moyen de $350\ 000 \pm 37\ \%$ lignes. Comme le paramètre de bruit semblait avoir un effet crucial sur les résultats, nous avons mené des expériences supplémentaires que nous détaillons maintenant.

Szita et Lőrincz ont exécuté chacune des trois expériences (pas de bruit, bruit constant et bruit décroissant) une seule fois pour des raisons de temps : leurs résultats expérimentaux ont nécessité un mois de calculs. Nous avons apporté un soin tout particulier à l'implémentation de notre simulateur de Tetris, notamment en termes d'op-

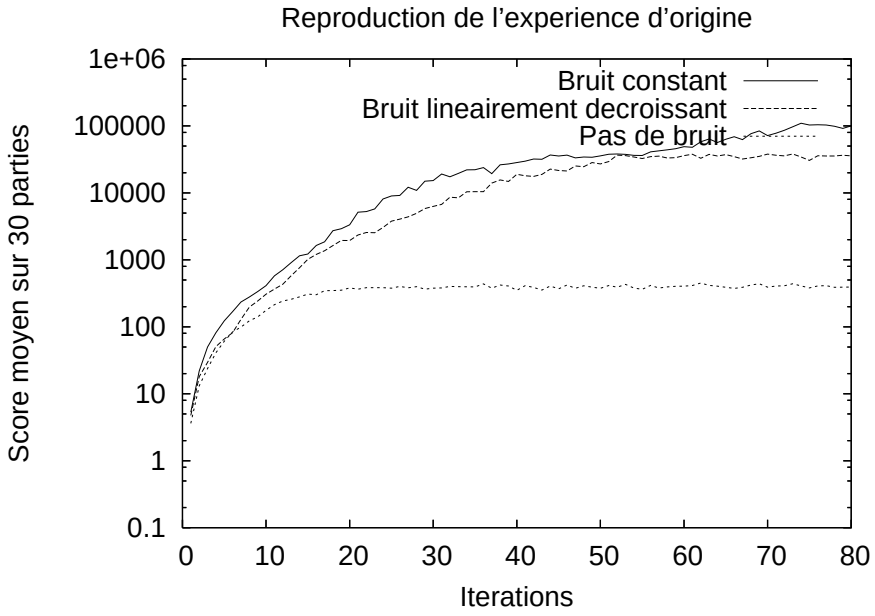


Figure 4. *Notre implémentation de l'expérience de Szita et Lőrincz . Chaque courbe représente la courbe d'apprentissage moyenne de 10 exécutions pour un type de bruit donné (en échelle logarithmique). Nous observons que l'ajout de bruit améliore significativement les performances. La meilleure des trois courbes d'apprentissage moyennes est celle qui correspond au bruit constant*

timisation, de manière à pouvoir reproduire leurs expériences plusieurs fois. En effet, nos premiers essais ont montré que plusieurs exécutions de la méthode d'entropie croisée avec les mêmes paramètres pouvaient donner des résultats très différents. Ainsi, nous avons décidé de lancer chacune des trois expériences de Szita et Lőrincz 10 fois. Avec notre implémentation optimisée de Tetris, cela a pris environ une semaine.

Les résultats que nous avons obtenus sont représentés aux figures 4 et 5. La figure 4 montre pour chaque type de bruit la courbe d'apprentissage moyenne de 10 exécutions. Nos résultats expérimentaux confirment l'observation de Szita et Lőrincz selon laquelle ajouter du bruit améliore significativement les résultats. Cependant, nous avons observé que la performance moyenne est meilleure avec le bruit constant. La figure 5 montre, pour chaque type de bruit, le détail des 10 exécutions. La meilleure performance est atteinte avec le bruit linéairement décroissant : une des 10 exécutions obtient un joueur artificiel qui réalise un score moyen de $240\,000 \pm 37\,\%$ lignes. Avec cet intervalle de confiance, nos résultats semblent cohérents avec ceux de Szita et Lőrincz ($350\,000 \pm 37\,\%$). L'examen de la figure 5 donne une meilleure idée sur le choix du bruit : les 10 exécutions avec le bruit constant atteignent toutes des performances

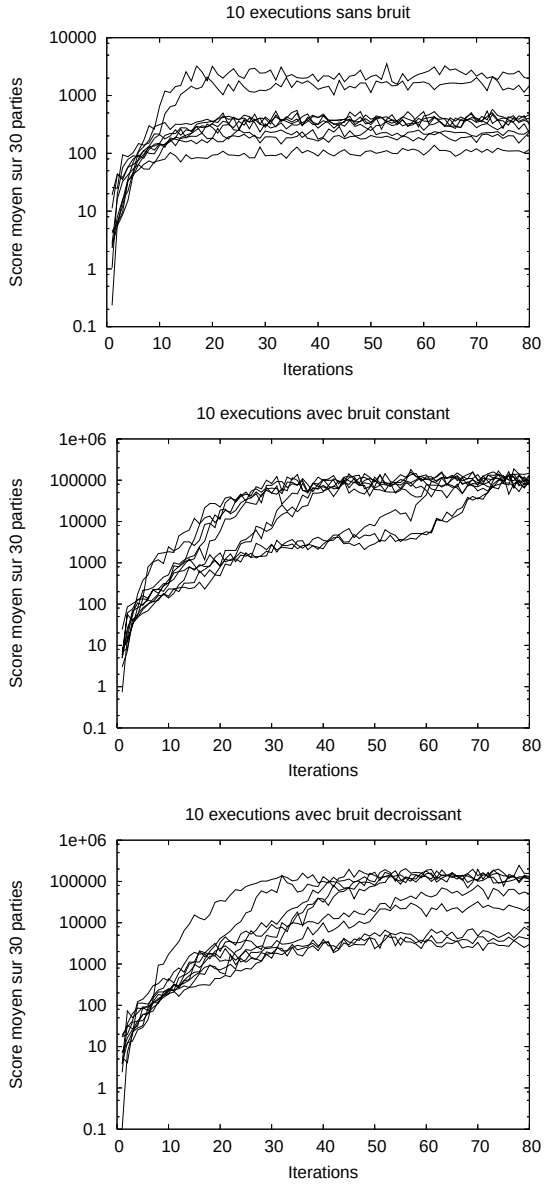


Figure 5. *Les 10 exécutions de chaque expérience de la figure 4 (en échelle logarithmique).* **Sans bruit :** *la courbe d'apprentissage se stabilise toujours après l'itération 20. Le score moyen atteint varie selon les exécutions (de 100 à 3 000 lignes).* **Bruit constant :** *les 10 exécutions atteignent des performances proches après convergence, entre 100 000 et 200 000 lignes.* **Bruit linéairement décroissant :** *les 10 exécutions atteignent des valeurs très différentes, de 5 000 à 250 000 lignes*

similaires après convergence ($100\ 000$ à $200\ 000 \pm 37\ \%$ lignes), alors qu'avec le bruit décroissant, les performances varient beaucoup entre plusieurs exécutions de la méthode d'entropie croisée. Cela est dû au fait que souvent, le bruit décroissant atteint zéro trop vite et l'algorithme converge avant d'avoir eu le temps de découvrir des bonnes solutions. Par conséquent, si on lance une seule exécution de la méthode d'entropie croisée (c'était le cas dans l'expérience originale de Szita et Lőrincz et ce sera le cas dans la prochaine section où nous construisons des joueurs artificiels qui jouent de très longues parties), le bruit constant est plus fiable, à moins de modifier la formule du bruit linéairement décroissant pour le faire diminuer moins vite. Cette conclusion semble aussi indiquer qu'à moins que l'implémentation de Tetris de Szita et Lőrincz diffère de la nôtre (voir la discussion à la section 3 sur l'influence significative de paramètres apparemment mineurs), le score de $350\ 000 \pm 37\ \%$ lignes par parties pourrait avoir été obtenu par chance dans la mesure où l'algorithme a été exécuté avec du bruit linéairement décroissant.

5. Notre joueur artificiel pour Tetris

Nous venons de voir que la méthode d'entropie croisée était un algorithme efficace pour optimiser les poids d'un ensemble de fonctions de base. Comme nous l'avons vu dans la description des travaux existants (section 2), il est également essentiel de choisir un ensemble de fonctions de base pertinent afin de capturer les aspects importants du jeu de Tetris. Ainsi, une approche naturelle, que nous appliquons dans cette section, est de considérer d'autres fonctions de base de Tetris que les fonctions simples de Bertsekas et Tsitsiklis (tableau 1).

Nous avons essayé plusieurs combinaisons de fonctions de base, dont celles de Dellacherie (tableau 2) puisqu'elles constituent le meilleur joueur artificiel pour Tetris à notre connaissance. Nous avons aussi introduit deux fonctions de base originales : la *profondeur des trous* et le *nombre de lignes avec trous*. La profondeur des trous indique à quelle distance de la surface du mur se trouvent les trous : c'est la somme du nombre de cellules pleines au-dessus de chaque trou. Le but de cette fonction est d'éviter d'enterrer trop profondément des trous. Notre seconde fonction de base originale compte le nombre de lignes ayant au moins un trou (deux trous sur la même ligne comptent pour un seul).

Nous avons exécuté la méthode d'entropie croisée sur le jeu de taille 10×20 dans les mêmes conditions que Szita et Lőrincz (Szita *et al.*, 2006) : nous avons commencé avec une gaussienne centrée à $\mu = (0, 0, \ldots, 0)$ avec variance $\sigma^2 = (100, 100, \ldots, 100)$, nous avons généré $N = 100$ vecteurs à chaque itération, et nous avons sélectionné les 10 meilleurs ($\rho = 10\ \%$). Conformément aux conclusions de la section précédente, chaque vecteur était évalué en jouant une seule partie, et nous avons choisi d'utiliser un bruit constant (avec la même amplitude que Szita et Lőrincz : $Z_t = 4$). Nous avons lancé l'algorithme avec 4 ensembles de fonctions de base différents : Dellacherie (D), Bertsekas + Dellacherie (BD), Dellacherie + Nous (DN), et Bertsekas + Dellacherie + Nous (BDN). Comme espéré, les performances obtenues

sont nettement meilleures que lorsqu'on se contente des fonctions de base de Bertsekas et Tsitsiklis. Comme les parties sont beaucoup plus longues, nous avons pu lancer une seule exécution pour chacun de ces 4 ensembles de fonctions de base. Bien que notre implémentation soit optimisée, en lançant les 4 expériences sur des machines différentes, ces expériences ont pris un mois.

La figure 6 fournit la courbe d'apprentissage pour chacun des 4 ensembles de fonctions de base. Comme dans l'expérience de Szita et Lőrincz , les courbes représentent le score moyen de 30 parties jouées avec le joueur artificiel moyen généré à la fin de chaque itération. La première observation que l'on peut faire est que nos deux fonctions de base originales ont un impact significatif sur les scores : les courbes correspondantes (les deux courbes en pointillés sur la figure 6) sont celles qui montent le plus haut. Nous observons également que si l'on supprime les fonctions de base de Bertsekas et Tsitsiklis (les expériences sans ces fonctions de base correspondent aux deux courbes épaisses), l'algorithme prend moins d'itérations pour converger (ce qui n'est pas surprenant puisqu'il y a moins de paramètres à optimiser) et atteint des scores similaires. Cela suggère qu'une fois que l'on a les fonctions de base de Dellacherie, celles de Bertsekas et Tsitsiklis ne donnent pas plus d'informations. On pourra enfin noter que les courbes de la figure 6 représentent le score moyen de seulement 30 parties, ce qui fait que l'intervalle de confiance correspondant est assez grand (± 37 %). Pour évaluer les joueurs artificiels plus précisément, nous avons sélectionné quelques joueurs artificiels pour chaque ensemble de fonctions de base (nous avons choisi quelques vecteurs de poids correspondant aux pics des courbes de la figure 6) et nous les avons fait jouer plus de parties. Le tableau 3 reporte, pour le meilleur joueur artificiel de chaque ensemble de fonctions de base, le score moyen de 100 parties sur un jeu de taille 10×20. L'intervalle de confiance correspondant est de ± 20 %. Nous avons également fait jouer ces mêmes joueurs artificiels sur un jeu de taille 10×16, pour avoir une borne inférieure sur le score qu'ils réaliseraient sur le Tetris original (voir section 3.2). Sur ce jeu de taille réduite, nous avons joué 1 600 parties (les parties étant plus courtes), ce qui donne un intervalle de confiance de 5 %.

Fonctions de base	**DN**	**BDN**	**D**	**BD**
Taille 10×20	35 000 000	35 000 000	17 000 000	20 000 000
Taille 10×16	910 000	910 000	530 000	660 000

Tableau 3. *Score moyen du meilleur joueur artificiel obtenu avec la méthode d'entropie croisée bruitée, pour chaque ensemble de fonctions de base. 100 parties ont été jouées en taille* 10×20 *(l'intervalle de confiance est* 20 %*) et 1 600 parties ont été jouées en* 10×16 *(l'intervalle de confiance est alors de* 5 %*). Les ensembles de fonctions de base sont représentés par leur première lettre : D pour Dellacherie, B pour Bertsekas et Tsitsiklis, N pour Nous. Les meilleurs résultats sont atteints avec les fonctions de base DN et BDN*

L'utilisation de la méthode d'entropie croisée pour optimiser les poids du joueur artificiel de Dellacherie est pertinente : par rapport aux poids originaux fixés à la main,

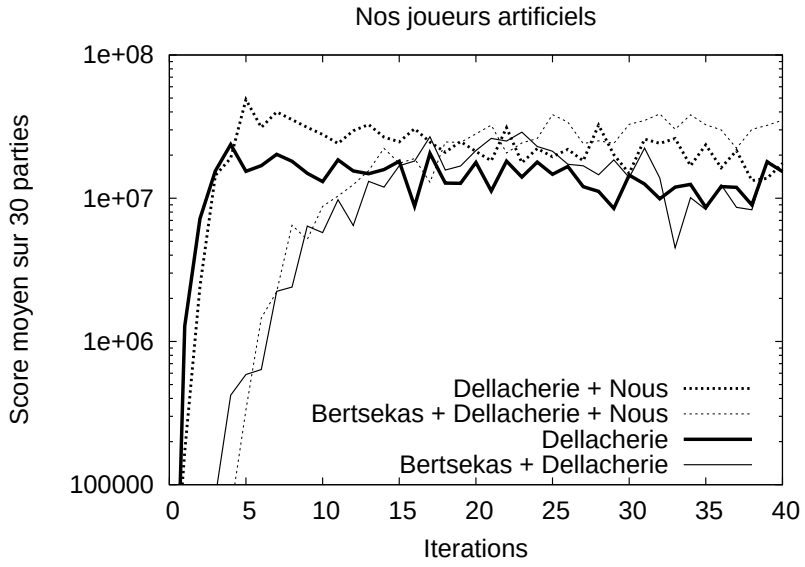


Figure 6. *Evolution du score moyen de 30 parties (en échelle logarithmique) avec la méthode d'entropie croisée bruitée, avec 4 ensembles de fonctions de base : Dellacherie, Dellacherie + Bertsekas, Dellacherie + Nous, Dellacherie + Bertsekas + Nous. Les deux courbes qui montent le plus haut sont celles où nos fonctions de base originales sont présentes (ce sont les deux courbes en pointillés). Lorsque les fonctions de base de Bertsekas et Tsitsiklis ne sont pas présentes (cela correspond aux deux courbes épaisses), l'algorithme converge beaucoup plus vite et les meilleures performances obtenues sont similaires*

Fonction de base	**Symbole**	**Poids**
Hauteur d'arrivée	l	-12.63
Erosion	e	6.60
Transitions de lignes	Δr	-9.22
Transitions de colonnes	Δc	-19.77
Trous	L	-13.08
Puits	W	-10.49
Profondeur des trous	D	-1.61
Lignes avec trous	R	-24.04

Tableau 4. *Les poids de notre joueur artificiel DN (Dellacherie + Nous). Score moyen :* $35\ 000\ 000 \pm 20\ \%$ *lignes en* 10×20 *et* $910\ 000 \pm 5\ \%$ *en* 10×16*. Voir section 5 et tableau 2 pour les définitions des fonctions de base*

les poids déterminés automatiquement améliorent les résultats par environ un facteur 3. Les meilleurs scores sont atteints avec les ensembles BDN et DN, qui réalisent des performances équivalentes : $35\,000\,000 \pm 20\,\%$ lignes sur le jeu de taille 10×20 et $910\,000 \pm 5\,\%$ lignes sur le jeu de taille 10×16. Ainsi, ces deux fonctions d'évaluation dépassent l'algorithme qui était jusqu'ici le meilleur à notre connaissance, celui de Dellacherie, qui réalise un score moyen de $5\,200\,000 \pm 20\,\%$ sur notre simulateur en 10×20 et $660\,000 \pm 27\,\%$ sur un simulateur du Tetris original (rappelons que jouer en 16×20 sur notre simulateur donne une borne inférieure assez pessimiste sur le résultat avec le Tetris original). Alors que le joueur artificiel BDN contient 28 fonctions de base, le joueur artificiel DN n'en possède que 8 : ce dernier est donc plus simple et plus rapide, et il peut ainsi être considéré comme meilleur. Nous donnons ses poids au tableau 4.

6. Conclusion et perspectives

Nous avons revisité l'application de la méthode d'entropie croisée au jeu de Tetris, proposée par Szita et Lőrincz (Szita *et al.*, 2006). En reproduisant 10 fois leur expérience d'origine, nous avons approfondi leur analyse expérimentale : en particulier, nous avons observé que le bruit constant semble plus fiable que le bruit linéairement décroissant. En utilisant la connaissance experte (les fonctions de base) de Dellacherie (Fahey, 2003) et deux fonctions de base originales, nous avons construit un joueur artificiel à une pièce qui donne à notre connaissance les meilleurs résultats à l'heure actuelle.

Pour approfondir nos expériences, il serait intéressant de modifier les formules de bruit (en effet, nos résultats indiquent que le bruit linéairement décroissant atteint trop vite zéro) ou en utilisant d'autres types de bruits, notamment un bruit avec décroissance géométrique. On pourrait aller plus loin en imaginant des fonctions de base plus complexes ou plus expressives. Une première direction naturelle serait d'exploiter d'autres fonctions de base de la littérature (par exemple celles de Xtris (Llima, 2005) ou Fahey (Fahey, 2003)) ou d'en inventer d'autres. Une piste de recherche particulièrement intéressante est le problème de sélectionner et de combiner automatiquement des fonctions de base simples de manière à construire d'autres fonctions de plus haut niveau. Par exemple, de telles combinaisons peuvent faire partie de l'espace de recherche, comme dans la récente approche de Programmation Génétique de Girgin et Preux (Girgin *et al.*, 2007).

D'une manière générale, un problème important concernant le jeu de Tetris est le temps d'exécution nécessaire pour jouer une partie, car les algorithmes ont besoin de jouer de nombreuses parties. C'est encore plus important lorsque les joueurs artificiels sont meilleurs, car les parties durent de plus en plus longtemps. Nous envisageons trois pistes pour évaluer un joueur artificiel en réduisant la durée d'une partie ou le nombre de parties jouées.

– Une première idée serait de lancer l'algorithme d'apprentissage sur un jeu de taille réduite. De cette manière, les parties sont plus courtes, on peut donc générer plus de vecteurs, les évaluer plus soigneusement et les itérations peuvent être plus rapides. Cependant, il n'est pas clair qu'un joueur artificiel obtenu en jouant sur une taille réduite joue ensuite bien sur le jeu standard (10×20). Nous avons effectué quelques expériences, et les performances des joueurs artificiels construits en jouant sur des plus petites grilles (comme 10×16) semblent donner des scores légèrement inférieurs à ceux des joueurs artificiels construits directement avec la grille standard.

– Nous avons vu que pour évaluer la qualité d'un joueur artificiel, il est préférable de jouer de nombreuses parties. Au lieu de jouer des parties aléatoires, une piste à explorer serait de jouer un petit ensemble de parties prédéterminées, avec des séquences de pièces générées à l'avance. Ainsi, avec cette méthode, on utiliserait les mêmes séquences de pièces pour comparer des joueurs artificiels différents. Dans la phase de sélection de l'algorithme d'entropie croisée, cela pourrait permettre de sélectionner les meilleurs échantillons de manière plus fiable dans la mesure où on utiliserait la même base de comparaison. Cela dit, il faut que les séquences de pièces prédéterminées soient suffisamment représentatives des parties possibles et il n'est pas clair qu'un joueur artificiel entraîné sur un ensemble restreint de séquences soit performant sur des parties qu'il n'a jamais rencontrées auparavant.

– Pour réduire le temps nécessaire pour évaluer un joueur artificiel, une idée prometteuse vient d'une conjecture de Fahey (Fahey, 2003), qui stipule que la durée d'une partie de Tetris (et, par conséquent, le nombre de lignes réalisées) peut être estimée à partir des premiers coups joués. En effet, considérons pour chaque hauteur h (c'est-à-dire $h = 0$ à 20) la *fréquence* de h, qui est la proportion du temps où la hauteur du mur a été exactement h pendant les n premiers coups. Avec un bon joueur artificiel, lorsque h est grand, la fréquence de h est faible puisque les murs hauts apparaissent peu souvent. Fahey a observé expérimentalement qu'avec son joueur artificiel, lorsque h augmente, la diminution de la fréquence de h est exponentielle. Nous avons fait des expériences qui confirment son observation pour nos joueurs artificiels. Par conséquent, si l'on estime les paramètres de cette distribution exponentielle (ils diffèrent pour chaque joueur artificiel) en effectuant une régression, on peut en déduire la fréquence pour $h = 21$, ce qui correspond à la fin de la partie. Pour un joueur artificiel donné, on pourrait ainsi estimer la durée moyenne d'une partie en jouant seulement n coups au lieu de faire une ou plusieurs parties. Cependant, nos premières observations indiquent qu'une telle méthode a une grande variance et manque donc de précision, même si l'on joue un grand nombre de coups (de l'ordre de $n = 1\,000\,000$ de coups).

Il reste à approfondir ces pistes afin de réduire le temps d'exécution de l'algorithme.

Enfin, nous travaillons actuellement sur l'algorithme CMA-ES (Hansen *et al.*, 2001) (Covariance Matrix Adaptation Evolution Strategy), qui peut être vu comme une alternative à la méthode d'entropie croisée. La différence principale est qu'avec CMA-ES, la distribution gaussienne qui génère des vecteurs de poids utilise une matrice de covariance. La distribution gaussienne utilisée dans la méthode d'entropie croisée correspond au cas d'une matrice de covariance diagonale. Sur la figure 3, les

axes des ellipses représentées sont toujours parallèles à un axe du repère. Avec l'algorithme CMA-ES, les axes des ellipses peuvent être dans n'importe quelle direction. Ainsi, l'espace de recherche de la distribution gaussienne est plus expressif et le processus de recherche peut être plus efficace. Nous examinons actuellement l'intérêt d'utiliser CMA-ES au lieu de la méthode d'entropie croisée.

7. Bibliographie

Bertsekas D., Tsitsiklis J., *Neurodynamic Programming*, Athena Scientific, 1996.

Böhm N., Kókai G., Mandl S., « An Evolutionary Approach to Tetris », *The Sixth Metaheuristics International Conference (MIC2005)*, 2005.

Burgiel H., « How to Lose At Tetris », *Mathematical Gazette*, vol. 81, p. 194-200, 1997.

de Boer P., Kroese D., Mannor S., Rubinstein R., « A tutorial on the cross-entropy method », *Annals of Operations Research*, vol. 1, n° 134, p. 19-67, 2004.

Demaine E. D., Hohenberger S., Liben-Nowell D., « Tetris is hard, even to approximate. », *Proc. 9th International Computing and Combinatorics Conference (COCOON 2003)*, p. 351-363, 2003.

Fahey C. P., « Tetris AI, Computer plays Tetris », 2003. `http://colinfahey.com/tetris/tetris.html`.

Farias V., van Roy B., *Tetris : A study of randomized constraint sampling*, Springer-Verlag, 2006.

Girgin S., Preux P., Feature Discovery in Reinforcement Learning using Genetic Programming, Technical Report n° RR-6358, INRIA, 2007.

Hansen N., Ostermeier A., « Completely Derandomized Self-Adaptation in Evolution Strategies », *Evolutionary Computation*, vol. 9, n° 2, p. 159-195, 2001.

Kakade S., « A natural policy gradient », *Advances in Neural Information Processing Systems (NIPS 14)*, p. 1531-1538, 2001.

Lagoudakis M. G., Parr R., Littman M. L., « Least-squares methods in reinforcement learning for control », *SETN '02 : Proceedings of the Second Hellenic Conference on AI*, Springer-Verlag, London, UK, p. 249-260, 2002.

Llima R. E., « Xtris readme », 2005. `http://www.iagora.com/~espel/xtris/README`.

Ramon J., Driessens K., « On the numeric stability of gaussian processes regression for relational reinforcement learning », *ICML-2004 Workshop on Relational Reinforcement Learning*, p. 10-14, 2004.

Sutton R., Barto A., *Reinforcement Learning, An introduction*, BradFord Book. The MIT Press, 1998.

Szita I., Lőrincz A., « Learning Tetris Using the Noisy Cross-Entropy Method », *Neural Computation*, vol. 18, n° 12, p. 2936-2941, 2006.

Tsitsiklis J. N., van Roy B., « Feature-Based Methods for Large Scale Dynamic Programming », *Machine Learning*, vol. 22, p. 59-94, 1996.